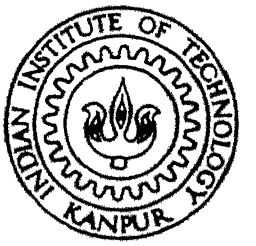
NETMON - A NETWORK MONITORING TOOL

by

P. V. NAGESWARA RAO



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

FEBRUARY 1997

NETMON - A NETWORK MONITORING TOOL

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by
P. V. NAGESWARA RAO

to the
Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur
February, 1997

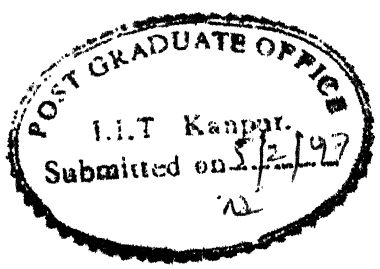
CSE-1997-M-RAC-NET

Certificate

Certified that the work contained in the thesis entitled "*NETMON - A NETWORK MONITORING TOOL*", by Mr.*P. V. NAGESWARA RAO*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Harish Karnick
Professor,
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

February, 1997



Acknowledgements

I would like to thank Dr. Harish Karnick, first for suggesting an exciting problem and then for encouraging and guiding me throughout the course of my thesis. My special thanks to him for making the timely submission of the thesis possible.

My thanks to Brahmaji, for being patient enough to hear to my problems and for acceding to my various requests. I would like to thank Mr. Nirmal Roberts and Mr. Aftab Alam for helping me during my thesis work. I appreciate the spirit of myriad software developers, who write excellent software and share it with others.

I would like to thank all my friends, specially the class of mtech95, for their co-operation and support. Hall-4 and its inmates will always occupy a special position in my memory. Special mention need to be made of Sarma, Dileep, Veluru, Raghuram, and Siva reddy with whom I had great time in IIT.

I would like to thank my parents and family members for providing me all the encouragement and support during the course of my studies.

Finally, I would like to thank the almighty for what I am today.

Abstract

Network Monitoring is very useful in diagnosing network related problems. Most of the existing network monitors give a snapshot of the network during the time of their execution. We have developed tools which monitor the network and store the traffic summaries over a period of time. This stored information aids in analysis of the traffic generated. Two tools, a *Protocol Analyzer* and an *NFS Analyzer* have been developed. The Protocol Analyzer captures the network traffic through a promiscuously configured Ethernet interface, and stores the protocol summaries. The NFS Analyzer monitors NFS traffic and stores filesystem load levels and other related information.

Contents

List of Figures

Chapter 1

Introduction

Network Management is very important in day-to-day running of computer systems in a networked environment. Network management needs information about the network like bandwidth utilization, protocol summaries, load on fileservers, and other related information. A lot of data has to be captured, stored and analyzed before attempting to perform any action on the system. Networks tend to grow very rapidly. If networks are to work efficiently inspite of rapid growth it is necessary to capture a variety of data which will not only allow us to manage static networks but also plan and manage the growth itself. As such, Network data capturing and management software becomes essential for proper functioning of the network.

Commercially available Network management software is quite good. But, in-house development of such software is desirable, since the functionality required at a particular site is very sensitive to the nature of the site and kinds of users it supports. For example, in a networked University environment containing many servers, workstations, PCs, and other special devices the proper distribution of user directories on various file systems can be an important parameter. The program can be fine tuned to the local requirements. Also, commercial software is quite expensive, whereas in-house development can be done without incurring much cost.

1.1 The Environment

In IIT Kanpur the Computer Centre is the main provider of computing resources. There is a campus-wide ethernet backbone connecting all the machines in the campus. The dominant operating system used is UNIX. TCP/IP is the protocol used for networking purposes.

The computer centre has a host of UNIX machines, workstations, terminals, and X Terminals. Sun NFS[SGK+85] is used for providing distributed filesystem services to the users. PC-NFS is used to connect PC's to the network. Internet connection is available to all the users through a gateway machine. Bridges, switches, routers and hubs connect the various ethernet segments. Plans are underway to upgrade the network, by installing an ATM backbone and deploying 100 Mbps fast ethernet in place of 10Mbps ethernet.

1.2 Motivation

The Computer Centre personnel are responsible for managing the network. The network is growing in size, and complexity. The staff is finding it difficult to cope with the complexity. Network Management Software is very much needed in these circumstances. SNMP(Simple Network Management Protocol) is a network management protocol, but it is not installed on many of the components in the network. No investments have been made on Network management software initially, as the network was small. But, later on the network grew, and with it arose the need for management software. In this thesis, an attempt has been made to develop a network monitoring tool, which will aid in network management.

1.3 Network Monitoring

Network monitoring tools monitor, analyze network traffic and suggest measures to be taken up for improving the Network performance. Often, the capability to see what is moving on the cable is quite useful. A trace of the packets flowing on the network gives an indication of the usage pattern of the network.

By observing the network traffic one can figure out facts like, which host is generating maximum amount of traffic, which host is receiving maximum amount of packets, etc. The dominant protocols in use can also be observed. In case of NFS traffic, one can determine the load on filesystems, find out the maximum NFS traffic generating clients, users, etc.

Often traces of this sort prove very helpful in making administrative decisions, like, moving file systems from one computer to another or moving files among filesystems, allocating home directories for new users, relocating existing user's login areas etc.

A network may further be divided into logical subnets, depending on the traffic patterns observed. For example, a set of hosts amongst whom the traffic is high, can be formed into a separate ethernet segment.

1.4 Existing Software

Lot of software already exists, to monitor network traffic and analyze it.

tcpdump[SMb], is a software to observe the network traffic. Packets, as they are flowing on the network, are captured and displayed. The source machine, destination machine, protocol, timestamp and other information is displayed. Command line options for specifying protocols to monitor, source machine, destination machine, etc., are available. *tcpdump* runs on almost all flavors of UNIX.

rpc.etherd and *traffic*, both available on Sun machines, provide a graphical display of the network traffic.

On the NFS front, **nfswatch**[CM] is a very useful program. This program shows the number of requests arriving onto filesystems, the number of requests emanating from client machines, users, and response time of the fileservers and other interesting details. Command line options for specifying source machine and/or destination machine, cycle time, total time of execution, etc., are available.

nfstrace[Bla92] is another interesting tool. This software produces trace of the NFS activity as well as a plausible set of corresponding system calls in the client machine.

Etherman[SFa], available from Curtin University, is an X11 based tool which displays a representation of real-time Ethernet communications. *Interman*[SFb], also from Curtin University, focuses on IP connectivity within a single segment.

Lot of related software is available on the Internet, in all forms, source, binary, free and also for sale.

1.5 Features of NETMON

NETMON is another network monitoring tool. It differs in one respect from the aforesaid monitors. NETMON stores the collected statistics for a user decided length of time, say 30 days, and can perform analysis on the stored data. Data is collected, summarized and organized on a daily basis. This storage of statistics makes possible answering queries of the type, "Which was the most accessed file system yesterday?", or, "Is there any increase/decrease of network traffic a day after the network reorganization has been done?", and similar ones.

NETMON has two components: Protocol Analyzer and NFS Analyzer.

Protocol Analyzer receives Network data promiscuously through a system 'network tap', and summarizes it into useful statistics, like Protocol wise breakup of the traffic, application wise breakup of the traffic, etc. Traffic between various sets of hosts, subnets, specified by the user, is also monitored. The collected statistics are stored for later reference by the user.

NFS Analyzer deals only with NFS traffic. NFS packets are captured, analyzed, and information like, total number of requests arriving on a filesystem, both reads and writes, number of requests emanating from a client machine, number of requests issued by a user, etc., are gathered.

1.6 Organization of the Thesis

Chapter 2 discusses basic concepts about protocols, protocol headers and network monitoring mechanisms. Some packet capturing mechanisms are also presented in this chapter.

Chapter 3 discusses the design of NETMON. Some design issues pertaining to NETMON are discussed in this chapter.

Chapter 4 discusses the implementation details of NETMON.

Chapter 5 discusses the experience with NETMON. Some observations of network usage are presented.

Chapter 6 concludes the thesis and describes future work that can be done.

Chapter 2

Background

Network Monitors capture packets on the network and extract useful information from them. To develop these monitors we need to know the structure of packets and protocols. Every protocol has its own header format. Network monitors parse protocol headers and generate statistics. Structure of ethernet packets and other protocol headers of interest to NETMON are discussed in this chapter.

2.1 Packet Structure

An ethernet packet size can vary from a minimum of 64 bytes to a maximum of 1518 bytes. Every packet on the network contains the ethernet header. Ethernet header consists of 48 bit source ethernet address, 48 bit destination ethernet address, and a 16 bit protocol field.

Next to the ethernet header, we have the network layer protocol header. IP is the network layer protocol in case of TCP/IP networks. Other protocol headers like Address Resolution Protocol (ARP) or Reverse Address Resolution Protocol (RARP) may be present in place of the IP header.

Next to the network layer protocol header, we have the transport layer protocol header. TCP and UDP are the transport layer protocols in case of TCP/IP networks. Internet Control and Message Protocol (ICMP) header may also exist in place of TCP/UDP headers.

Next to the transport layer protocol header, we have the application layer protocol header. Some application level protocols run on UDP, while some run on TCP. For example ftp, telnet, http run on TCP, while RPC, TFTP run on UDP.

Figure 1 gives the protocol hierarchy.

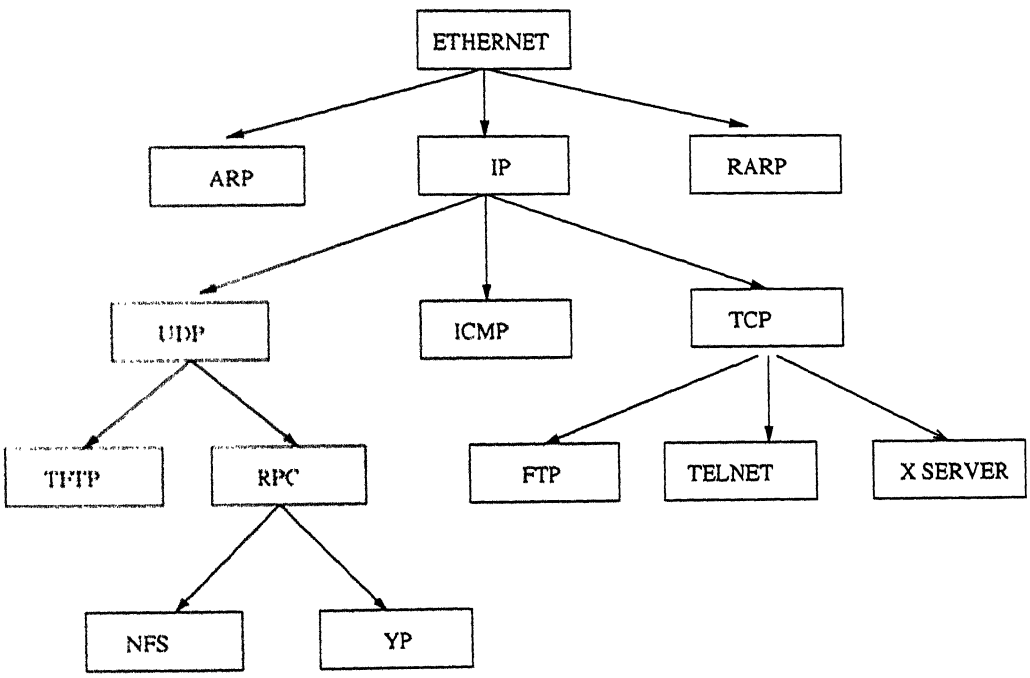


Figure 1: Protocol Hierarchy

2.1.1 IP header

The IP header contains a number of fields. Of interest to NETMON are

- source IP address
- destination IP address
- length of IP header
- length of data
- protocol

The protocol field specifies protocol of the Transport layer, say TCP, UDP or ICMP.

2.1.2 TCP header

The TCP header contains a number of fields. Of interest to NETMON are

- source port
- destination port

Source port and destination port contain TCP port numbers that identify application programs at the ends of the connection. A port is the abstraction that transport protocols use to distinguish among multiple destinations (application programs) within a given host computer. [Com95]

2.1.3 UDP header

The UDP header contains various fields. Fields of interest to NETMON are

- UDP source port
- UDP destination port

The source port and destination port fields contain the 16 bit UDP protocol port numbers used to demultiplex datagrams among the processes waiting to receive them. The source port is optional. When used, it specifies the port to which replies should be sent; if not used, it should be zero.

2.1.4 RPC header

RPC stands for Remote Procedure Call. RPC runs on top of UDP. There are two types of RPC packets.

- RPC call
- RPC reply

RPC call

Every RPC call packet contains a call header. There are a number of fields in the header. Of these only a few are of interest to NETMON. They are

- program number. This field contains the RPC program number.
- version number. This field contains the version number of the program.
- procedure number. This field contains the remote procedure number, which is to be executed.
- message id. This field is used to match RPC replies with calls.

RPC reply

Every RPC reply packet contains the reply header. The fields in the reply header of interest to NETMON are

- message id. This field is common to both call and reply headers. This field is used to match responses with requests.

RPC replies do not carry the RPC program number. Message id field has to be used to match replies with calls.

2.1.5 Well known ports

The Operating System reserves some ports for standard services. Both TCP and UDP have defined a group of well-known ports. For example, every TCP/IP implementation that supports FTP, the File Transfer Protocol, assigns port 21 to it[Ste94]. TFTP, Trivial File Transfer protocol, is assigned the UDP port of 69. Some other well known ports are

- telnet - 23
- smtp - 25
- finger - 79
- bootps - 67

2.2 Network File System

In this section a brief description of the Network File System is given. The Sun Network Programming manuals [Sun88] provide more details.

The Sun Network Filesystem (NFS) protocol provides transparent remote access to shared files across networks. The NFS protocol is designed to be portable across different machines, operating systems, network architectures, and transport protocols[Inc89].

An NFS network consists of servers and clients. A particular machine can be a client or a server or both. Server machines export filesystems for access by clients. Client machines mount remote filesystems in their local hierarchy for accessing them.

Remote files and directories are referred to by a file handle. File handle formats vary from implementation to implementation. File handles are opaque to the client machine and are only understood by the server.

The interface between client and server is defined in terms of 17 RPC operations. There are operations to read or write files(read, write), create or remove files (create, remove), create or remove directories(mkdir, rmdir), get file attributes(getattr), get filesystem attributes(statfs) and for other functions.

Client machines cache the remote files for better performance. Cache coherence is maintained by the client machine by checking the last write time of the file at the server, before using the cached data.

NFS is built on top of RPC. An RPC transaction consists of a call message from the client to the server and a reply message from the server to the client. RPC calls are transmitted using the UDP/IP protocols. The call message contains program number, version number and procedure number. There is also a unique transaction identifier which is included in the reply message to enable the client to match the reply with its call. The data in these messages is encoded in an "external data representation" (XDR), which provides a machine-independent standard for byte order, etc.

NFS is a stateless protocol. No client site information is stored at the server. The absence of any client site information at the server makes crash recovery simple.

2.3 Capturing Packets

User processes can capture packets on the network in a *system dependent* manner. For this, the ethernet interface has to be configured to operate in promiscuous mode. This setting can only be done by the superuser. On some systems, ordinary users are allowed to listen on ethernet interfaces, while on some other systems, only root can listen to the packets. For example, on DEC OSF/1, ordinary users can snoop on to the network, while on Sun machines, only the root is allowed to do so.

In this section, we discuss ways to capture packets on DEC OSF/1 and Sun OS systems. Every system has its own way of providing data link level access to users.

2.3.1 PacketFilter

In DEC OSF/1 systems, packetfilter[MRA87] provides access to network packets. This section gives a brief introduction to packetfilter. Further details can be obtained from the manual.

The packet filter pseudo device driver provides a raw interface to Ethernets and similar network data link layers[Dig94]. The packet filter driver is kernel-resident

code provided by DEC OSF/1 operating system. The driver appears to applications as a set of character special files, one for each open packet filter application.

For opening these special files, the Operating System provides the *pfopen* library routine. Associated with each open instance of a packet filter special file is a user settable packet filter "program", that is used to select which incoming packets are delivered by that packet filter special file.

Reads from these files return the next packet from a queue of packets that have matched the filter. Writes to these files, transmit packets on the network, with each write operation generating exactly one packet.

The packet filters treat the entire packet, including headers, as uninterpreted data. The user must supply the headers for transmitted packets (although the system makes sure that the source address is correct) and the headers of received packets are delivered to the user. The packet filter mechanism does not know anything about the data portion of the packets it sends and receives.

The packet filter supports a variety of different ethernet data-link levels say 10Mbps ethernet, FDDI.

2.3.2 Network Interface Tap

Sun's Network Interface Tap(NIT) is the facility provided by Sun machines to provide link-level network access. This facility comprises of a set of modules which collectively provide facilities for constructing applications that require link-level network access. They are explained below :

- *nit_if* is the component that provides access to the network interfaces.
- *nit_pf* is the packet filtering module which can be used to specify filters for accepting packets.
- *nit_buf* is the module that buffers incoming messages, thereby, reducing the number of system calls and associated overhead required to read and process them individually.

  NIT clients mix and match these components, based on their particular requirements. Examples of applications, which use NIT include *rarpd*, which

is a user-level implementation of the Reverse ARP protocol, and *etherfind* which is a network monitoring and trouble-shooting program available on Sun machines.

Chapter 3

Design of NETMON

NETMON has four components in it : the *protocol analyzer* processes packets and classifies them according to the protocol used, the *nfs analyzer* analyzes the NFS traffic, the *storage unit* stores the collected statistics and retrieves them on demand, and the user interface interacts with the user. The overall design of NETMON and design of the individual components is described in this chapter.

3.1 Architecture of NETMON

The packet capture unit is the interface with the Operating System. This interface keeps supplying NETMON, the packets flowing on the network in real time. The *protocol analyzer* takes each packet, strips the protocol headers, classifies the packet according to the protocol used at various levels, say at network layer level, at transport layer level and at application layer level. The corresponding counters are updated on each occurrence of the packet.

The *nfs analyzer* picks up NFS request packets only, i.e. requests going from client machine to the fileserver. As NFS is based on RPC/XDR, the packets have to be reconstructed. This is needed because data in XDR format has to be converted to local machine format. After reconstructing the packet, the

server filesystem, client machine, and the user on whose behalf this request has been issued, are determined. The counters associated with the filesystem, client machine and user are then updated.

The *storage unit* stores the statistics collected by the analyzer programs, the nfs analyzer and the protocol analyzer. These statistics are retrieved when the user requests them.

The *user interface* is a character-based interface and is quite simple. Queries to obtain traffic details over a period of time are supported. Traffic details for a day, an hour or for a 5 minute interval can be obtained. Traffic summaries for the last n days, n hours, or n 5 minute intervals are also available. In case of protocol analyzer, additional operations, to add/delete hosts and subnets to monitor, to generate a list of hosts, subnets being monitored and to perform other miscellaneous functions are provided.

Figure 2 depicts the Architecture of NETMON.

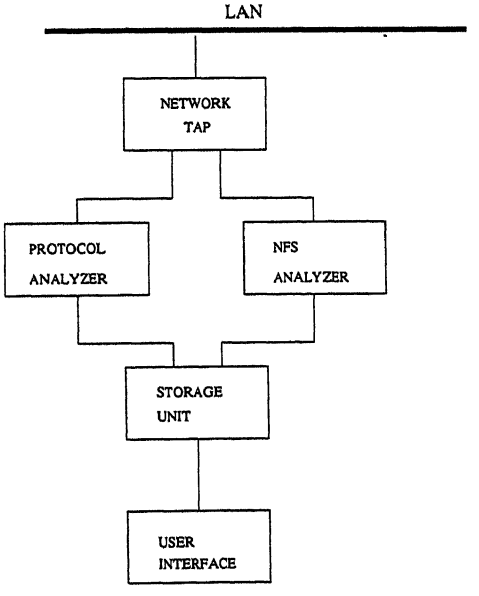


Figure 2: Architecture of NETMON

3.2 Protocol Analyzer

The **protocol analyzer** runs through the ethernet packets, stripping the protocol headers and extracts the protocols and applications associated with the packet.

An ethernet packet is taken and the network layer protocol is found out from the *'protocol'* field of the ethernet header.

Next, the ethernet header is stripped off. The packet might be an ARP, RARP, IP packet or others. For ARP and RARP, no further processing is done as they do not contain any further encapsulation.

After extracting an IP packet, we observe the *'protocol'* field. This field may point to TCP, UDP or ICMP protocol. For ICMP packets, no further processing is required. The icmp packet counter is incremented by one on every occurrence of an ICMP packet.

After extracting a TCP packet, the source port and destination port fields in the TCP header are examined. Some standard applications have well-known ports associated with them. For example, FTP runs on 21, telnet on 23, and SMTP(mail) on 25. Depending upon the source port or the destination port, the application is found out. If the application cannot be determined in this way, the packet is treated as belonging to miscellaneous TCP applications. The application level protocols currently recognized by NETMON are ftp, telnet, X, http, smtp and nntp.

For a UDP packet, the processing is similar. We again observe the source port and destination port and try to figure out the application. If we can't, the packet is tested for a RPC packet. In case of a RPC call, the RPC program number is examined. NFS, YP, MOUNT, PCNFSD are some of the applications running on RPC which are recognized by NETMON. In case of a RPC reply, no further processing is required. All the packets are grouped into the *rpc replies* category. Remaining packets are treated as miscellaneous UDP applications.

The protocol analyzer, analyzes the overall network traffic. Options are also available to monitor traffic between

- a particular host and set of other hosts.
- a particular host and a set of subnets.
- a subnet and a set of other subnets.

3.3 NFS Analyzer

NFS analyzer, as the name suggests, analyzes the NFS traffic going onto all the exported filesystems and generates statistics on filesytem usage levels. NFS request packets on the network are captured. The packet is reconstructed, i.e. data in XDR format is converted to the local machine's format. From the reconstructed packet, the server filesystem, the client machine and the user on whose behalf the request has been issued are found out. The corresponding counters are then updated.

The NFS request packets arriving onto a filesystem are classified as NFS_READ or NFS_WRITE, depending on, whether the the execution of the NFS procedure contained in the packet, results in a read operation or write operation at the fileserver.

3.4 Storage Unit

This unit is responsible for storing the statistics collected by the protocol analyzer and the nfs analyzer.

The statistics are collected by the analyzer programs continuously. After every five minute interval, the statistics collected in the interval are stored and counters reset. Again, after every one hour interval, the statistics collected in the preceding hour are summarized and stored. Similarly, after every twenty

four hour interval (a day), the statistics generated during that day are summarized and stored. In this way, data collected is stored in units of day, hour and five minute intervals.

Currently, statistics summaries for a maximum period of 30 days are stored by NETMON.

3.5 User Interface

The User Interface is quite simple. Only a character based interface is provided. The analyzer program act as server, while the user interface program acts as the client. UDP is the protocol employed for the client-server interaction in case of protocol analyzer, while the nfs analyzer uses TCP for the same purpose.

The client program accepts input from the user, and sends these requests to the analyzer program for processing. The server processes the requests and sends responses to the client, i.e. the user interface program. The user interface program then displays the results obtained.

The following queries are supported by both the programs, the nfs analyzer and the protocol analyzer.

- Get statistics collected in the last *'n'*th day interval.
- Get statistics collected in the last *'n'*th hour interval.
- Get statistics collected in the last *'n'*th five minute interval.

Similarly, queries to observe summaries over the last 'n' days, 'n' hours and 'n' five minute intervals are available.

In case of protocol analyzer, commands, to add a host(or subnet) to monitor, to delete a host(or subnet) being monitored, are provided. A command to obtain the startup time of the analyzer is also provided.

3.6 Design Decisions

The *protocol analyzer* and the *nfs analyzer* have been allowed to exist as two separate programs. This is because, the nfs analyzer is specific to NFS, and is useful only when there is a lot of NFS traffic on the segment. The protocol analyzer is of a very generic nature and can be used on any ethernet segment.

The User Interface is quite simple. Only a character based interface is provided. The functionality to be built into the system is given more importance than the user interface.

Protocol analyzer monitors traffic between a particular host and a set of subnets, or between a subnet and a set of other subnets. By subnet, we mean a set of IP addresses, but not exactly the set of hosts in the domain. For example, all hosts with address prefix of 144.16.163 are treated to be part of the subnet 144.16.163.*. This has been done to make implementation simpler. Domains and the hosts in them, have to be defined for exact working of these options.

The minimum monitoring interval has been fixed at five minutes. This software is mainly used for performing retrospective analysis of traffic at macro intervals of time, say on a daily basis. As such, the minimum monitoring interval need not be at a micro level, say every second. Micro level interval updates are used by software like *Etherman*[SFa], which display real-time Ethernet communications instantaneously.

Chapter 4

Implementation Details

The *protocol analyzer* and the *nfs analyzer* exist as independent programs. The statistics storage part is again similar for both of them. Only what is stored differs, not how it is stored. The User Interface of both the programs is identical, except for the presence of a few additional options for the protocol analyzer. This chapter discusses the implementation details of NETMON.

4.1 Protocol Analyzer

The protocol analyzer is implemented as a separate module. Some major steps in the execution of the program are explained below.

4.1.1 Initialization

This is the first step performed in the program. Basic initialization chores are performed in this step. A socket is created and set up for asynchronous I/O. This is done for accepting user's requests from time to time. UDP port number 7419 is used by the protocol analyzer, while the nfs analyzer uses TCP port number 7219 for the same purpose. The ethernet interface is then properly

configured to accept packets in promiscuous mode by a series of ioctl calls. Finally, various data structures are initialized or allocated memory as needed.

4.1.2 Input Files

The program has two configuration files, *hosts.monitor* and *mapfile*. The former is used to specify the list of hosts and subnets to monitor. The later is needed by the nfs analyzer and is a feature originally present in *nfswatch*. The two input files and their contents are discussed below.

Monitor File

NETMON monitors the overall network traffic in the ethernet segment. In addition, it also allows one to monitor the following types of traffic.

- Traffic 'arriving to', and 'generated from' a host. This type of monitor is referred to as a 'H' monitor hereafter.
- Traffic flowing between a particular host and a set of other hosts. This type of monitor is referred to as a 'HH' monitor hereafter.
- Traffic flowing between a particular host and a set of other subnets. This type of monitor is referred to as a 'HN' monitor hereafter.
- Traffic flowing between a particular subnet and a set of other subnets. This type of monitor is referred to as a 'NN' monitor hereafter.

These monitors are setup at startup time by specifying them in the *hosts.monitor* file. Entries in the file are explained below.

A 'H' monitor is set up by specifying the hostname or IP address of the host, one per line. For example,

yamuna

monitors traffic 'proceeding to' or 'arriving from' *yamuna*.

A 'HH' monitor is set up by specifying the hostname of interest, followed by number of secondary hosts, and then their names/IP addresses. Again one HH entry is specified on one line. For example,

csex1 2 csealpha1 csealpha2

monitors traffic flowing between csex1 and csealpha1, csealpha2 combined. Number 2 indicates the two auxiliary hosts of interest to *csex1*.

A 'HN' monitor is set up by specifying the hostname of interest, followed by the number of subnets, and then the subnet addresses. For example,

*csesparc1 1 144.16.163.**

monitors traffic flowing between *csesparc1* and hosts with a IP prefix of 144.16.163.

A 'NN' monitor is set up by specifying the subnet of interest, followed by the number of secondary subnets, and then their subnet addresses. For example,

144.16.162. 1 144.16.163.**

monitors traffic flowing between hosts with IP prefix of 144.16.162 and hosts with IP prefix of 144.16.163.

A sample input file is given in the Appendix.

Mapfile

The *mapfile* is an accoutrement with the *nfswatch* program. As we have used *nfswatch* code in our program, our program too needs it.

The *mapfile* contains a list of device names and file system names (one pair per line). The program reads pairs of the file system device specifications (the file server's name/address, and major number, minor number of the device) and the proper names of the file systems from mapfile. Each line should contain a string representing what *nfswatch* would normally print, and then separated from that by whitespace, the name that is preferred. For example:

csealpha1(8,1030) csealpha1:/1d2

csesun1(7,6) csesun1:/var/spool/mail

The "preferred names" are printed while displaying filesystem names, rather than the less readable filesystem device specifications.

4.1.3 Parsing Packets

From the raw ethernet packet, the higher level protocols are found out by gradually stripping the headers and noting protocol types. Protocols and the id's corresponding to them can be found in the system include files. For example, TCP's number is 6, while UDP's is 17. By observing the protocol id's, the network level, and transport level protocols are found out.

The application level protocols are found out in a slightly different way. The port numbers, obtained from TCP/UDP header are examined, and if they correspond to any of the well known ports, we get to know the application from the port number. Some applications using well known ports are, ftp - 21, telnet - 23, smtp - 25, tftp - 69.

UDP packets may have to be explored further, as they may be encapsulating RPC packets. RPC packets are of two types, call and reply. Every RPC call packet contains the program number in it. Some common RPC programs are NFS, YP, MOUNT etc. RPC reply packets do not contain any program number, so they are all bundled into *rpc replies* category.

The program maintains counters in long unsigned integers. Structures are used to define counters.

The following structure is used to store the packet count of protocols at the network layer level.

```
typedef struct  {
        LN_COUNTER ip_cnt; /* IP packet counter */
        LN_COUNTER arp_cnt; /* ARP packet counter */
        LN_COUNTER rarp_cnt; /* RARP packet counter */
        LN_COUNTER nw_others; /* others counter  */
} NW_LEVEL;
```

The following structure is used to store the packet count of protocols at the transport layer level.

```
typedef struct {
        LN_COUNTER udp_cnt; /* UDP packet counter */
        LN_COUNTER tcp_cnt; /* TCP packet counter */
        LN_COUNTER icmp_cnt; /* ICMP packet counter */
        LN_COUNTER tp_others; /* others */
} TP_LEVEL ;
```

The following structure is used to store the packet count of protocols at the application level.

```
typedef struct {
        LN_COUNTER rpc_call; /* RPC calls */
        LN_COUNTER rpc_cnt;  /* RPC replies
        LN_COUNTER nfs_cnt; /* NFS calls */
        LN_COUNTER telnet_cnt; /* TELNET packet counter */
        LN_COUNTER ftp_cnt; /* FTP packet counter */
        LN_COUNTER smtp_cnt; /* SMTP packet counter */
        LN_COUNTER www_cnt; /* HTTP packet counter */
        LN_COUNTER nntp_cnt; /* NNTP packet counter */
        LN_COUNTER xserver_cnt; /* X traffic counter */
        LN_COUNTER tftp_cnt;     /* TFTP counter */
        LN_COUNTER yp_cnt;       /* Yellow pages traffic */
        LN_COUNTER mount_cnt;    /* NFS mount protocol count */
        LN_COUNTER pcnfsd_cnt;   /* PC-NFSD traffic count */
        LN_COUNTER tcp_app_others; /* other tcp apps */
        LN_COUNTER udp_app_others; /* other udp apps */
} APP_LEVEL ;
```

The Protocol Analyzer uses the following basic cell for storing traffic details observed in a unit of time. This unit may be a day, or an hour, or 5 minutes.

```
typedef struct {
        LN_COUNTER raw_cnt;     /* raw ethernet pkt's cnt */
        LN_COUNTER bdcst_cnt; /* ethenet broadcasts     */
        NW_LEVEL nw_lvl;
        TP_LEVEL tp_lvl;
        APP_LEVEL app_lvl;
        int from_secs;   /* start time of the monitoring */
        int to_secs;     /* end time of the monitoring */
} CELL ;
```

4.2 NFS Analyzer

NFS Analyzer is the second module of NETMON. It monitors NFS traffic only. It is based on *nfswatch*. nfswatch is freeware and is available from many sites on the net. nfswatch code has been liberally used in this module. NFS Analyzer works as follows :

Initialization for nfs analyzer is identical to protocol analyzer, in the sense that the ethernet interface is configured in promiscuous mode, socket is setup and data structures initialized.

NFS Analyzer processes only NFS requests, not responses. NFS packets are picked up for scrutiny, by observing the transport level protocol(which is UDP in case of NFS), and the RPC program number. Most implementations use, port number 2049 as NFS port, but this is not a standard practice.

The NFS packet's data will be in XDR format when it is captured from the network. This data is *deserialized* using XDR decode routines. These routines are provided by nfswatch and are directly used.

Once deserialized, we extract the NFS procedure and classify it as a NFS_READ or NFS_WRITE operation.

The following NFS procedures are classified as NFS_READ operations, as their execution at the file server results in read operations.

- *getattr()* Returns file attributes. This is like a stat call.
- *lookup()* Returns file handle and attributes for a file in a directory.
- *readlink()* Returns the string associated with the symbolic link file.
- *read()* Returns data from a file.
- *readdir()* Returns directory entries.
- *statfs()* Returns filesystem information.

The following NFS procedures are classified as NFS_WRITE operations as their execution at the file server results in write operations.

- *setattr()* Sets the attributes of a file.
- *write()* Writes specified number of bytes in the file.
- *create()* Creates a new file and returns the file handle and attributes.
- *remove()* Removes a file from a directory.
- *rename()* Renames a file.
- *link()* Creates a link to a file.
- *symlink()* Creates a symbolic link.
- *mkdir()* Creates a new directory
- *rmdir()* Removes an empty directory.

Thus each NFS packet is classified as a read or write operation. The client machine issuing the call, and the user on whose behalf the call has been issued are extracted from the packet. The server's address is extracted and the filehandle is decoded. Decoding NFS filehandles is a tricky issue, as various implementations use their own proprietary formats. Again, filehandle decoding part is directly taken from nfswatch.

For maintaining the count of 'NFS requests' arriving onto a filesytem, the program has the following data structure.

```
typedef struct _fs_ {
        my_devt nc_dev; /* device numbers of file sys   */
        my_fsid nc_fsid; /* for ''learning'' file systems  */
        ipaddrt nc_ipaddr; /* keep track of server address */
        char    *nc_name; /* name of file system          */

        Counter cnt_nfsread; /*count file system read ops.  */
        Counter cnt_nfswrite; /*count file system write ops. */
        Counter nc_interval; /* total requests this interval */
        Counter nc_proc[MAXNFSPROC]; /* nfs proc counters */
        struct _fs_ *fs_next; /* link field */

} NFSCounter;
```

The purpose of each field is specified in comments.

For maintaining the count of NFS requests emanating from a client machine the program has the following data structure.

```
typedef struct _cl_ {
        ipaddrt cl_ipaddr;              /* client IP address      */
        char    *cl_name;               /* name of client system  */
        Counter cl_interval;            /* requests this interva  */
        struct  _cl_ *cl_next;          /* hash chain link        */
} ClientCounter;
```

For maintaining the count of NFS requests generated by a user the program has the following data structure.

```
typedef struct _ac_ {
        long    ac_uid;                 /* authorization type     */
        char    *ac_name;               /* name of user id        */
        Counter ac_interval;            /* requests this interval */
```

```
        struct  _ac_    *ac_next;       /* hash chain link        */
} AuthCounter;
```

The NFS Analyzer uses the following basic cell for storing traffic details observed in a unit of time. This unit may be a day, or an hour, or a 5 mt interval.

```
typedef struct _cell_ {
        NFSCounter *ptr_fs_counters;    /* file system counters */
        ClientCounter *ptr_cl_counters; /* client counters */
        AuthCounter *ptr_auth_counters; /* Authentication counters */
        int cnt_fsentries;    /* No. of file system counters in cell */
        int cnt_clentries;    /* No. of client counters in this cell */
        int cnt_authentries;  /* No. of auth counters in this cell   */

        int from_secs; /*start time of monitoring interval*/
        int to_secs;     /*End time of monitoring interval */
} NFS_CELL;
```

The statistics storage is akin to the one employed by the protocol analyzer.

4.3 Statistics Storage and User Interface

Statistics Storage unit is identical to both the analyzers. Only, what is stored changes, not how it is stored. There are 3 kinds of lists, *days list*, *hrs list* and *mts list*. These lists store the basic cells. A cell is the basic data storage unit and is different for both the analyzers. The cells are given above.

The data, as captured from the network is first collected into *current* counters. After every five minute interval, data in the *current* counters is added to the tail of the *mts list* and the *current* counters are reset. After every one hour interval, the statistics of the last, 12 five minute intervals are added up to

obtain the hour cell. This cell is then added at the tail of the *hrs list*. After every 24 hour interval, the statistics of the last 24 hours are added up to obtain the day cell. This cell is then added at the the tail of the *days list*. The lists are implemented as circular queues.

In case of NFS Analyzer, daily traffic is stored on disk file, while everything is in memory for the protocol analyzer.

User Interface

User Interface is implemented using the client-server paradigm. The User Interface program is the client while the analyzer program is the server. Client program accepts input from the user, and sends it to the server. The server processes the request and sends back the response. The client programs display these responses. A list of the commands is given in the Appendix.

4.4 Implementation Issues

HTTP daemon usually runs on the well known port number 80. Due to some reasons the http daemon is now running on some other port on the HTTP proxy. Therefore, traffic 'arriving from' or 'proceeding to' the proxy on the new httpd port is treated as http traffic.

The X server port is also not a well known port. We are assuming the port number 6000 as the X port. This is a valid assumption as most implementations use the same port for the X Server.

Only the number of packets is counted, not the number of bytes in the packet. Both counts are desirable. Packet counts give the physical number of packets on the wire, while byte count gives the bandwidth utilization levels.

Chapter 5

Observations

The Protocol Analyzer and the NFS Analyzer have ben tested thoroughly. Long traces of file system traffic and protocol summaries have been obtained. In this chapter we discuss the observations that can be made after examining the collected statistics.

5.1 Experience with nfsanalyzer

The nfs analyzer has been run, both on the CSE segment and the CC segment. The nfs traffic on both these segments is observed. In this section, the program's output is analyzed.

5.1.1 FileSystem load levels

The nfs analyzer gives the traffic arriving on each exported filesystem in the segment. Sample output follows :

```
Start time = Sat Jan  4 16:07:01 1997
End time   = Sat Jan 11 16:07:01 1997
```

```
No. of active file systems in this interval = 45

LIST OF FILE SYSTEMS

SER:FS csesparc1:/dos     read = 2572045 write = 554609 total = 3126654
SER:FS csealpha1:/1d2     read =  484416 write = 242600 total =  727016
SER:FS csealpha1:/usr     read =  155409 write =      0 total =  155409
SER:FS godavari:Unknown   read =   67402 write =  42626 total =  110028
SER:FS arjun(7,2127360)   read =   21788 write =   2126 total =   23914
                                    .
                                    .
                                    .
```

The duration interval is 7 days. The start time gives the time when the monitoring interval started, and end time gives the time it ended. The next line gives the total number of filesystems onto which NFS requests have come. Next, we have a list of filesystems, and no. of read requests, write requests and total requests arriving onto them. The output is sorted so that busy filesystems come at the top of the list.

The numbers are of little importance when taken in an absolute sense. Only a relative comparison would be meaningful. The */dos* filesystem on *csesparc1* is the busiest filesystem. This filesystem contains DOS files, which are mounted on the PC's through nfs. It also has system areas for some diskless workstations in the lab.

/1d2 filesystem on csealpha1 is the next busy filesystem. This filesystem contains home directories of most of the users.

/usr on csealpha1 contains read only binaries. Therefore, no writes are performed on this filesystem.

godavari is a Linux PC, and it's filehandle format is not recognized by the program. For recognizing new filehandle formats, new code has to be added.

arjun(7,2127360) is the filesystem on arjun, which is being accessed from machines in the CSE segment. The numbers in braces indicate the major and minor numbers of the filesystem. The major and minor numbers are printed, when no corresponding filesystem name is specified in the mapfile.

5.1.2 Client machines list

The nfs analyzer produces the list of client machines generating NFS requests. A sample client's list looks as follows :

```
No. of active client machines in the interval = 71

LIST OF CLIENT MACHINES

Client csealpha3        requests = 2025728
Client csealpha2        requests = 1937223
Client rpt              requests = 1552453
Client pc35             requests = 1030454
Client 144.16.163.171   requests =  145516
                        .
                        .
                        .
```

There are 71 client machines which have issued NFS calls in the timeperiod of interest. The machine name, and the number of calls it has generated are given one per line. The output is sorted, so that most active client machine stands at the top of the list.

5.1.3 Users list

The nfs analyzer also generates the list of users, on whose behalf the NFS calls have been issued. A sample output looks as follows :

```
No. of active users in the interval = 240

LIST OF USERS
USER root       requests = 2065999
USER rpt        requests = 1559981
USER nobody     requests =  924740
USER naidu      requests =  718124
USER #9138      requests =   44075
                .
                .
                .
```

There are 240 users issuing NFS calls in the monitored interval. The login-id of the user and number of requests made are given, one per line. Again, the output is sorted, so that most active users top the list. *root* is the id used by all the system processes. *nobody* is a login id associated with PCNFS implementation. The uid is given if the login-id can't be obtained from the local passwd server.

Observing the NFS traffic for a long amount of time, say a month, helps us to find out any surges, or drops in traffic after a reorganization of the filesystems, home directories, etc. The effect of a reorganization exercise can thus be observed with the nfs analyzer.

5.2 Experience with Protocol Analyzer

The protocol analyzer has also been run on both CSE and CC segments. Observations made after examining the traffic are presented in this section. Overall network traffic in a sample five-minute interval is as follows :

```
GENERIC N/W TRAFFIC ANALYSIS
Start time = Tue Feb  4 17:56:09 1997
 End time = Tue Feb  4 18:01:09 1997
 Monitoring Duration (hh:mm:ss) 00:05:00

raw cnt                =     71224 ( 100.00 )
bdcst cnt              =        66 ( 0.09 )

ip cnt                 =     70988 ( 99.67 )
arp cnt                =       143 ( 0.20 )
rarp cnt               =         0 ( 0.00 )
nw others cnt          =        16 ( 0.02 )

udp cnt                =     39813 ( 55.90 )
tcp cnt                =     31147 ( 43.73 )
icmp cnt               =        28 ( 0.04 )
tpothers cnt           =         0 ( 0.00 )

telnet cnt             =      7007 ( 9.84 )
ftp cnt                =       143 ( 0.20 )
smtp cnt               =        49 ( 0.07 )
www cnt                =       127 ( 0.18 )
xserver cnt            =     23225 ( 32.61 )
nntp cnt               =         0 ( 0.00 )
tcp other app          =       596 ( 0.84 )

rpc_call cnt           =      2137 ( 3.00 )
rpc_reply cnt          =     17178 ( 24.12 )
nfs calls              =     16542 ( 23.23 )
yp calls               =       523 ( 0.73 )
tftp cnt               =         1 ( 0.00 )
```

```
mount calls          =        18 ( 0.03 )
pcnfsd calls         =         0 ( 0.00 )
udp other app        =      3414 ( 4.79 )

dropped              =      5886
oveflows(cumul)      =         0
```

The figures displayed are the absolute number of packets on the network in the given interval. The numbers in braces indicate the percentage of that component in the total traffic. The difference between the *raw cnt* and sum of *ip cnt*, *bdcst cnt*, *arp cnt*, *rarp cnt* and *nw others cnt*, gives a count of fragmented packets. In the above case, we have 11 such packets.

From the output, it is evident that, UDP is the dominant protocol(56%), closely followed by TCP(44%). Of the UDP traffic, NFS is again the dominant protocol. We can see that nfs calls account for 23%, and rpc replies account to 24%. It should be noted that nfs replies count is bundled up with other rpc replies. Little amount of YP (yellow pages) traffic is also observed.

In case of TCP traffic, X traffic is the most dominant one, followed by telnet, HTTP and others.

dropped gives a count of packets dropped since the start of monitoring.

Similarly, composition of traffic, 'incoming' and 'outgoing' for a host, flowing between a host and subnet(s), flowing between a subnet and other subnet(s) can be obtained.

Chapter 6

Conclusions and Further Work

Primarily our attempt in this thesis has been to develop a tool for network monitoring. The data collected by the monitor is to be used by the system administrators. A protocol analyzer for analyzing the overall network traffic is built. This tool gives protocol wise breakup of network traffic. It recognizes most of the commonly used protocols and applications, such as ftp, telnet, X, http, etc.

An NFS Analyzer for analyzing the NFS traffic is also developed. It is modeled after *nfswatch*, except for the way it stores statistics. This tool generates the list of filesystems being accessed, list of client machines issuing NFS calls, and users responsible for these calls, and their load levels. These statistics are expected to be used by the system administrators in allocating filesystems to machines, users home directories to filesystems, and for other administrative decisions.

Both the tools have been developed on DEC OSF/1, using the packetfilter mechanism for packet capture. The software can be made to run on other systems as well, by suitably changing the packet capture unit. libpcap[SMa], a system independent API, for user level packet capture can be used for this purpose.

Further Work

This project is the first step towards building a network management software for the IITK Computer Centre. As such, there is a lot of scope for further work.

Currently, NETMON only collects statistics. The interpretation and analysis of the data generated, has to be done by the user. It would be very useful, if the analysis part is also done by the software. The ideal situation, wherein the software collects statistics, and analyzes them thoroughly, and suggests corrective actions for performance improvement, is desirable. Lot of work need to be done in this direction.

The protocol analyzer is far from complete. RPC traffic, especially RPC replies, need to be further classified. Right now, all RPC replies are bundled into one category. Newer protocols, like IPV6 are to be recognized. Also, a capability, wherein the program can detect new protocols automatically and store information about them is desirable.

On the NFS front, more work can be done. The procedure wise breakup of the NFS traffic is to be generated. Next, there is no way to correlate among the observed three entities, namely filesystems, client machines and users. This feature enhancement would do a lot of good to the program. Also, design and implementation for constructing user profiles, i.e., about their usage patterns of files, filesystems, and resources has to be done.

Both the above tools have to be upgraded to work on 100 Mbps network as well, since the Computer Centre is planning to install these networks soon.

Lastly, the User Interface needs a big face lift. Providing a graphical interface, and displaying the collected statistics graphically, by means of charts, graphs would make the program easy to use.

Appendix A

User Manual

NETMON can be used by system administrators interested in observing network traffic. Programmers also can use this tool for developing applications which need network traffic.

NETMON consists of two components, the protocol analyzer and the nfs analyzer. The protocol analyzer captures all the ethernet traffic and produces protocol wise breakup of the network traffic. The nfs analyzer captures only NFS traffic and generates statistics on filesystem usage levels, etc.

Installation The distribution comes in the form of a tar file, *netmon.tar*. Extract the files from the archive to see two directories created, PROT_PROG and NFS_PROG. The PROT_PROG directory contains the protocol analyzer program, while NFS_PROG contains the nfs analyzer program. Create the executables by typing 'make' in both the directories. The executables, *nfsspy* and *nfsreport* should be created in the NFS_PROG directory and *protspy* and *protreport* created in the PROT_PROG directory.

Running

nfsspy puts the ethernet interface in promiscuous mode and captures NFS request packets only. The statistics collected are stored by the program. *nfsreport* accepts inputs from the user and obtains the traffic summaries from nfsspy. Both communicate through a TCP connection.

protspy is similar to nfsspy, except that it captures all the network traffic and stores summaries of the observed traffic. *protreport* should be used to fetch the data collected by protspy.

Edit the configuration files *mapfile* and *hosts.monitor*, to suit your environment. The former is used by the nfs analyzer and the latter by the protocol analyzer. Run the two programs, protspy and nfsspy in the background. These programs keep collecting statistics till they are terminated. You can view the collected statistics through the user interface programs, nfsreport or protreport.

User Interface

The Interface is as follows :

First, one has to enter a single character command. The program then prompts for arguments. The arguments are then to be entered in the appropriate format. The results are then printed on the screen or to a file. The commands and the arguments they take are listed below.

Statistics Retrieval Commands

- **m** Display statistics collected in the last *nth* five minute interval.
  Arg : *n*. A number between 1 to 60, specifying the interval of interest.
- **h** Display statistics collected in the last *nth* hour interval.
  Arg : *n*. The hour number of interest, ranging between 1 and 24.
- **d** Display statistics collected in the last *nth* day interval.
  Arg : *n*. The day number of interest, ranging between 1 and 30.
- **M** Display summary statistics over the last *n* five minute intervals.
  Arg : *n*. The duration of interest, ranging from 1 to 60.
- **H** Display summary statistics over the last *n* hours of interest.
  Arg : *n*. Number of hours over which the statistic summaries are to be generated. Any number ranging from 1 to 24.

- **D** Display summary statistics collected over the last n days.

  Arg : n. The number of days over which the statistic summaries are to be generated. Any number ranging between 1 and 30.

Miscellaneous Commands

- **c** Executes miscellaneous functions.

  Arguments :

  * *uptime.* Shows the time the daemon has started and, the time elapsed since its start.
  * *print.* Prints the results of the commands to a file *res.out*, rather than on to the screen.
  * *unprint.* Clears the 'print to file' setting.

- **?** Help command. Displays help information about the program.
- **x** Leads to termination of the daemon program.
- **q** Quits the Interface program.

Protocol Analyzer Specific Commands The Protocol Analyzer has some additional commands. They are

- **a** Add a Host or subnet for monitoring.

  Arguments :

  * Enter hostname/IP address of the machine for *host* monitoring.
  * Enter hostname/IP address, number of auxiliary hosts and the name/address of auxiliary hosts for *host to host* monitoring.
  * Enter hostname/IP address, number of auxiliary subnets of interest, and the subnet addresses, for *host to subnet* monitoring.
  * Enter subnet address, number of auxiliary subnets of interest, and the subnet addresses, for *subnet to subnet* monitoring.

  The format of the lines entered here should be identical to the one specified in the *hosts.monitor* file. Please refer Appendix B for sample file.

- n Lists all the hosts, subnets being monitored currently, and lets the user select an entity for viewing statistics.

  Arg : *code*. A number given as 'code' for one of the entities in the generated list.

- z Deletes an entity being monitored. Displays a list of hosts, subnets being monitored currently, and lets the user delete an entity.

  Arg : *code*. A number given as 'code' for one of the entities in the generated list.

Appendix B

An Example Monitors File

```
# File Name : hosts.monitor
# Contains list of nodes to be monitored
# Date : 2/10/96.

cs1
yamuna
pc35
pc20
csp1
# Now some host to host monitoring.
csex1 3 cd1 pc11 csesparc2
csealpha1 1 csealpha2
#Now for some host to net monitoring.
csp1 1 144.16.163.*
csp2 1 144.16.163.*
#Now for some net to net monitoring.
144.16.162.* 1 144.16.163.*
144.16.161.* 1 144.16.162.*
```

Appendix C

An Example FileSystem Specification file

```
csealpha1(8,6) csealpha1:/usr
csealpha1(8,7) csealpha1:/1d1
csealpha1(8,1030) csealpha1:/1d2
csealpha1(8,1031) csealpha1:/1d3
csealpha2(8,7) csealpha2:/2d1
csealpha2(8,2054) csealpha2:/2d2
csealpha2(8,2055) csealpha2:/2d3
csealpha3(8,2048) csealpha3:/3d3
csesun1(7,6) csesun1:/var/spool/mail
csesun1(7,3) csesun1:/export/root
csesun1(7,14) csesun1:/usr
csesparc1(7,15) csesparc1:/4u1
csesparc1(7,14) csesparc1:/4u2
csesparc1(7,16) csesparc1:/dos
```

Bibliography

[Bla92] Matt Blaze. NFS tracing by passive network monitoring. In *USENIX Conference Proceedings*, pages 333–344, San Francisco, CA, Winter 1992. USENIX.

[CM] Dave Curry and Jeff Mogul. Nfswatch 4.2 ftp://coast.cs.purdue.edu/pub/tools/unix/nfswatch/nfswatch4.2.tar.gz.

[Com95] Douglas E. Comer. *Internetworking with TCP/IP Vol. I.* Prentice Hall India Private Ltd., second edition, 1995.

[Dig94] Digital Equipment Corporation. *Unix Programming Manual*, Feb 1994.

[GC94] Tim Kindberg George Couloris, Jean Dollimore. *Distributed Systems Concepts and Design.* Addison Wesley Publishing House, second edition, 1994.

[Hew89] Hewlett Packard Inc. *Using and Administering NFS Services*, Jan 1989.

[Inc89] Sun MicroSystems Inc. Nfs: Network file system protocol specification, rfc 1094, March 1989.

[MRA87] Jeffrey C. Mogul, Richard F. Rashid, and Michael J. Accetta. The packet filter: An efficient mechanism for user level network code. In *11th ACM Symposium on OS Principles*, pages 39–51. ACM, 1987.

[NRG] Lawrence Berkeley National Laboratory Network Research Group. Arpwatch 1.1, ftp://ftp.ee.lbl.gov/arpwatch.tar.z.

[SFa] Mike Schulze and Craig Farrell. Etherman 1.1, ftp://ftp.cs.curtin.edu.au.

[SFb] Mike Schulze and Craig Farrell. Interman 1.1, ftp://ftp.cs.curtin.edu.au.

[SGK+85] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the sun network filesystem. In *USENIX Conference Proceedings*, pages 119–130, Portland, OR, Summer 1985. USENIX.

[SMa] Van Jacobson Steve McCanne, Craig Leres. Libpcap 0.2.1, ftp://ftp.ee.lbl.gov/libpcap.tar.z.

[SMb] Van Jacobson Steve McCanne, Craig Leres. Tcpdump 3.0.2, ftp://ftp.ee.lbl.gov/tcpdump-*.tar.z.

[Ste94] Richard W. Stevens. *Unix Network Programming.* Prentice Hall India Private Ltd., 1994.

[Sun88] Sun MicroSystems Inc. *Sun Network Programming Manual*, May 1988.
